

WBS no. 1030013-02 Client: 0953 Order date 01/21/2003 Priority: N

Your order number: AUS8-02-1238/Vitello/M. Dawkins

Source: Chongbo Kwahakhoe Nonmunji B (J. of KISS, Korea Information Science Society, pt. B, Software and Applications)

vol. 22 issue: 10 Date: Oct. 1995 pp. 1404-15

AU: Giho Choi et al.

TI: Providing Multiple BLOB Types for Efficient Storage of Multimedia Data

Journal of KISS(B): Software and Applications

VOL. 22, NO. 10, OCTOBER 1995

Databases

- A Far-Future-Using(FFU) Buffer Replacement Algorithm for Continuous Media Taeck-Geun Kwon 1395
Sukho Lee
Providing Multiple BLOB Types for Efficient Storage of Multimedia Data Giho Choi 1404
Eunji Kang
Hyunchul Kang

Software Engineering

- Design of the Map Class and Executing VDM Formal Specifications Gi Hwon Kwon 1416
Software Testing Algorithm Using Data Flow Analysis Techniques Hyun-Suk Hwang 1425
Man-Gon Park
Design of an Expert System for Software Quality Evaluation Kum-sook We 1434
Keum-suk Lee

Artificial Intelligence

- Stereo Image Matching Based on Straight-line Feature for Positioning Jong-Woong Choe 1445
Kyung-Haeng Lee
Tae-Kyun Kim

Programming Languages

- Graph Reduction Machine Using Activation Records Ahn, Joonseon 1459
Han, Taisook
Transforming Imperative Programs into Dataflow Program Graphs Yoonhee Nah 1470
Heunghwan Kim
Sangyong Han

Korean Language Processing

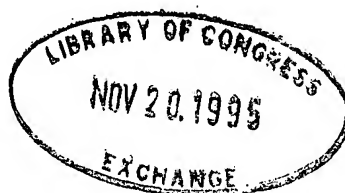
- Morphological Analysis of Korean Irregular Verbs Using Syllable Characteristics Kang, Seung-Shik 1480
Integration of Phoneme Recognition with Morphological Analysis
for Spoken Korean Processing Kyunghee Kim 1488
Geunbae Lee
Jong-Hyeok Lee

The Korea Information Science Society

제 22 권 제 10 호 1995년 10월

Journal of KISS(B): Software and Applications

VOL. 22, NO. 10, OCTOBER 1995



데이터베이스

- 연속 매체를 위한 FFU 버퍼 재배치 알고리즘 권택근, 이석호 1395
멀티미디어 데이터의 효율적 저장을 위한 BLOB 타입의 다양화 최기호, 강은지, 강현철 1404

소프트웨어 공학

- Map 클래스의 설계 및 VDM 정형적 명세의 실행 권기현 1416
데이터 흐름 분석 기법을 이용한 소프트웨어 테스트 알고리즘 황현숙, 박만곤 1425
소프트웨어 품질 평가를 위한 전문가 시스템의 설계 위금숙, 이금석 1434

인공지능

- 위치결정을 위한 직선특징기반 스테레오 영상의 정합 최중웅, 이경행, 김태균 1445

프로그래밍 언어

- 액티베이션 레코드를 이용하는 그래프 축약기계 안준선, 한태숙 1459
명령형 프로그램의 데이터플로우 프로그램 그래프로의 변환 나윤희, 김홍환, 한상영 1470

한국어정보처리

- 음절 특성을 이용한 한국어 불규칙 용언의 형태소 분석 강승식 1480
한국어 음성 언어 처리를 위한 음소 단위 인식과 형태소 분석의 결합 김경희, 이근배, 이종혁 1488



사단 법인 한국정보과학회

멀티미디어 데이터의 효율적 저장을 위한 BLOB 타입의 다양화

(Providing Multiple BLOB Types for Efficient Storage of
Multimedia Data)

최 기 호 * 강 은 지 ** 강 현 철 ***

(Giho Choi) (Eunji Kang) (Hyunchul Kang)

요 약 멀티미디어 데이터를 저장하기 위한 BLOB(Binary Large Object)에 관한 기존의 연구는 범용성을 갖는 단일 BLOB 타입의 구현을 목표로 하고 있다. 즉, 크기가 무제한인 BLOB을 저장할 수 있어야 하고, 임의의 바이트 범위(byte range)에 대한 부분 검색 및 변경 연산을 효율적으로 지원할 수 있는 BLOB 타입들이 제안되었다. 본 논문에서는, 멀티미디어 DBMS에서 단일 BLOB 타입을 제공하는 것보다 멀티미디어 응용의 특성을 고려하여 읽기 전용(read only) BLOB, 첨부 전용(append only) BLOB, 범용 BLOB 등 특성화된 다양한 BLOB 타입들을 제공하는 것이 더 효율적임을 제안한다. 사용자는 저장하고자 하는 멀티미디어 데이터별로 응용 상 특성을 고려하여 가장 적절한 BLOB 타입을 선택함으로써 효율성을 제고할 수 있다.

Abstract In the previous works on the BLOB(Binary Large Object) to store multimedia data, the goal has been to realize a singleton BLOB type which is general-purpose. Such a BLOB type is supposed to store any BLOB of unlimited size and to efficiently process partial scan/update operations against any byte range of the BLOB. In this paper, we suggest that it is more efficient for a multimedia DBMS to provide multiple BLOB types such as a read only BLOB, an append only BLOB, a general-purpose BLOB, and so on than to deal with all BLOBs with a singleton BLOB type. Each of these specialized BLOB types could be designed to efficiently meet some particular (not all) requirements of the data in multimedia applications. As such, users can enhance efficiency in storing multimedia data by choosing an appropriate BLOB type that best fits the data in question.

1. 서 론

CAD/CAM, CASE, CIM, office information system, 그리고 GIS와 같은 공학 응용들에서는 숫자, 문자 등의 전통적인 데이터 외에 텍스트, 이미지, 그래픽스, 비디오, 애니메이션, 사운드, 보이스 등의 멀티미디어 데이터를 저장하고 관리해야 한다. 숫자나 문자는 정형화된 짧은(formatted and short) 데이터인데 반하

여 멀티미디어 데이터는 비정형의 긴(unformatted and long) 데이터이다. [1]에서는 멀티미디어 데이터를 비롯하여 긴 데이터 또는 큰 데이터를 저장하기 위한 새로운 데이터 타입으로서 BLOB(Binary Large Object)이라는 용어를 사용하였다. 특히, 문자 텍스트로 구성된 BLOB을 텍스트 BLOB, 이진 데이터 스트림(binary data stream)으로 구성된 BLOB을 바이트 BLOB으로 구분하기도 하였다. BLOB이란 용어는 초기 연구자들이 사용한 긴 필드(long field)라는 용어, 그리고 이후 보다 더 포괄적인 의미로 사용된 큰 객체(large object)라는 용어 등과 혼용되면서 비정형의 긴/큰 멀티미디어 데이터를 저장하기 위한 데이터 타입의 의미를 갖게 되었다[1].

본 논문은 관계 데이터베이스 시스템에서 멀티미디어

* 이 연구는 '94년도 한국과학재단 연구비 지원에 의한 결과임(과제번호: 941-0900-004-1)

* 준 회원 : 한국통신기술

** 준 회원 : 중앙대학교 컴퓨터공학과

*** 통신회원 : 중앙대학교 컴퓨터공학과 교수

논문접수 : 1995년 5월 24일

심사완료 : 1995년 8월 22일

대
법
는
BL
을
적
을
[3]
(1)
(2)
일정
본
공간
하고
본
입을
어
타입
자는
성을
써
효
본
BLOI
BLOI
어 DE
한 BL
가를
BLOB
술한다
제를

2.

BLC
효율적
와 (2)
수 있다
본다. 모
장 구조
하여 기

1) ISO의
struct
LOB에
Large
Object

데이터를 저장하기 위하여 BLOB 타입을 지원하는 방법에 관한 것이다. 기존의 BLOB 연구들은 범용성을 갖는 단일 BLOB 타입의 구현을 목표로 진전되어 왔다. BLOB 타입의 범용성이란, 임의의 크기를 갖는 BLOB을 저장할 수 있고, 다양한 형태의 연산들을 모두 효율적으로 지원하며, BLOB을 관리하기 위한 제반 요건들을 효율적으로 만족할 수 있음을 의미한다. 예를 들어, [3]에서는 범용 BLOB 타입이 갖추어야 할 요건으로서 (1) 크기가 무제한인 BLOB을 저장할 수 있어야 하고, (2) BLOB의 전체에 대한 연산 뿐만 아니라, BLOB의 일정 바이트 범위(byte range)에 대한 여러 종류의 부분 연산들을 모두 효율적으로 지원해야 하고, (3) 저장 공간의 사용 효율이 100%에 가까워야 할 것 등을 언급하고 있다.

본 논문에서는, 멀티미디어 DBMS가 단일 BLOB 타입을 제공하는 것보다, 서로 다른 요건을 갖는 멀티미디어 데이터의 특성을 반영할 수 있도록 다양한 BLOB 타입들을 제공하는 것이 더 효율적임을 제안한다. 사용자는 저장하고자 하는 멀티미디어 데이터의 응용상 특성을 고려하여 가장 적절한 BLOB 타입을 선택함으로써 효율성을 제고할 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 기존의 BLOB 관련 연구를 기술하고, 3절에서는 다양한 BLOB 타입의 필요성을 제시하고, 4절에서는 멀티미디어 DBMS가 단일 BLOB 타입을 제공하는 것보다 다양한 BLOB 타입들을 제공하는 것이 얼마나 더 효율적인가를 기존 관계 DBMS 저장 시스템 상에서 3가지 BLOB 타입의 구현을 통한 실험으로 평가한 결과를 기술한다. 마지막으로 5절에서 결론을 맺고 향후 연구 과제를 기술한다.

2. 관련 연구

BLOB 타입에 관한 기존 연구는 크게 (1) BLOB의 효율적인 저장 구조를 제안하는 연구 [4][5][6][7][3]와 (2) BLOB 타입의 성능 평가 연구 [6][8]로 나눌 수 있다²⁾. 본 절에서는 이들 기존 연구에 대하여 알아본다. 먼저 2.1절에서 기존에 제안된 BLOB 타입의 저장 구조를 살펴보고, 2.2절에서 이들의 성능 평가에 대하여 기술한다.

2.1 BLOB 타입의 저장 구조

BLOB이라는 용어 [1]가 사용되기 이전부터 비정형의 긴 필드에 대한 연구는 1980년대 초부터 시작되었다. System R을 확장하여 긴 필드를 지원하도록 하는 연구 [4]가 있었고, WiSS [5], EXODUS [6], Starburst [7], 그리고 EOS [3] 등에서 긴 데이터 또는 큰 객체를 지원하기 위한 연구가 지속적으로 수행되었다. 이들 중 System R 및 WiSS의 긴 데이터 저장 구조는 디스크 블록에 기반을 둔 것이고, EXODUS, Starburst, 그리고 EOS의 큰 객체 저장 구조는 세그먼트(segment)에 기반을 둔 것이다. 세그먼트란 디스크 공간 상의 물리적으로 인접한 디스크 블록 (또는 페이지) 들로 구성된 것으로서 디스크 관리자 (또는 긴 필드/큰 객체 관리자)가 디스크 공간을 할당 및 반환하는 단위이다. 본 논문에서는 세그먼트 기반의 저장 구조를 갖는 BLOB 타입을 연구 대상으로 한다. 따라서 본 절에서는 EXODUS, Starburst, 그리고 EOS에서 제시된 세그먼트 기반의 BLOB 저장 구조를 살펴본다.

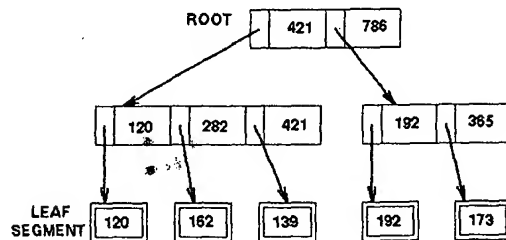


그림 1 EXODUS의 큰 저장 객체 예 [6]

2.1.1 EXODUS의 큰 저장 객체

EXODUS에서는 비정형의 긴 데이터를 큰 저장 객체 (large storage object)라고 부른다 [6]. 큰 저장 객체는 객체 내의 바이트 오프셋(byte offset)을 키(key)로 하는 B⁺ 트리 인덱스와 이 B⁺ 트리의 잎 노드로서 큰 저장 객체를 저장하고 있는 세그먼트들의 집합으로 구성된다. 이들 세그먼트를 제외한 B⁺ 트리의 각 노드는 그 자식 노드에 대한 여러개의 (바이트 수, 페이지 식별자) 쌍을 저장하고 있다. 여기서 바이트 수는 해당 자식 노

1) ISO의 SQL 관련 표준 사양 제정에 있어서는 LOB(unstructured Large Object)이라는 용어가 사용되고 있다 [2]. LOB에는 BLOB(Binary Large Object), CLOB(Character Large Object), 그리고 NCLOB(National Character Large Object) 등이 있다.

2) BLOB은 이들 연구가 수행된 프로토타입 데이터베이스 시스템에서 뿐만 아니라, Informix-OnLine, Oracle version 7, Empress, Sybase, UniSQL 등의 관계 및 객체 지향 모델에 기반을 둔 상용 데이터베이스 시스템에서도 널리 지원되고 있다 [9]. 국내에서 개발된 관계 DBMS인 한국전자통신연구소의 바다, 삼성전자의 CODA, 대우통신의 한바다 등도 모두 BLOB을 지원하고 있다.

드를 뿌리 노드로 하는 서브 트리가 저장할 수 있는 최대 바이트 수를 나타낸다. 일 노드 세그먼트는 저장 효율을 언제나 50% 이상으로 유지한다.

그림 1은 큰 저장 객체의 예를 보여 주고 있다 [6]. B* 트리의 뿌리 노드의 왼쪽 자식 노드는 1에서 421번째 바이트까지 저장하고, 오른쪽 자식 노드는 422에서 786번째 바이트까지 저장함을 나타낸다. 만약 큰 저장 객체의 713번째 바이트부터 접근하고자 할 경우, 가장 오른쪽 일 페이지의 100번째 바이트부터 접근하면 된다. 왜냐하면 $421 + 192 + 100 = 713$ 이기 때문이다 [6].

2.1.2 Starburst의 긴 필드

Starburst는 비정형의 긴 데이터를 저장하기 위해서 긴 필드 관리자(long field manager)를 제공하고 있다 [7]. 긴 필드의 저장 구조는, 긴 필드를 여러개의 서로 다른 크기 세그먼트들에 나누어 저장하고, 릴레이션 튜플의 해당 필드에 긴 필드 서술자(long field descriptor)를 저장함으로써 이들 세그먼트들을 가리키도록 하는 것이다. 즉, 긴 필드 서술자는 세그먼트들에 대한 디렉토리이다. 긴 필드 서술자의 최대 크기는 튜플 내의 한 필드값으로 저장되기 때문에 255 바이트로 제한된다.

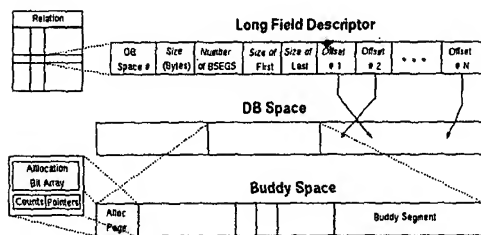


그림 2 Starburst의 긴 필드 저장 구조 [7]

긴 필드의 저장 구조를 나타내면 그림 2와 같다 [7]. 긴 필드를 저장하고 있는 세그먼트를 Starburst에서는 버디 세그먼트(buddy segment)라고 부른다. 이는 긴 필드 관리자가 디스크 공간을 관리하기 위해 버디 시스템(buddy system)을 사용하기 때문이다. 버디 세그먼트는 데이터베이스 공간(DB space)이라 명명되는 디스크 상의 한 부분인 버디 공간(buddy space)으로부터 할당되는데, 할당되는 세그먼트의 크기는 두배씩 증가된다. 예를 들어, 페이지의 크기가 1K 바이트라고 할 때, 첫번째 세그먼트의 크기는 1 페이지, 두번째 세그먼트의 크기는 2 페이지, 세번째 세그먼트의 크기는 4 페이지,

다음은 8 페이지, 다음은 16 페이지와 같은 방식으로 최대 크기 세그먼트인 2,048 페이지 (2M 바이트)까지 증가하여, 이후의 세그먼트는 2,048 페이지를 단위로 할당된다.

2.1.3 EOS의 큰 동적 객체

AT&T의 EOS(Experimental Object Store)는 DBMS 구현을 위한 저장 시스템이다. EOS의 큰 동적 객체(large dynamic object) [3]는 EXODUS의 큰 저장 객체 및 Starburst의 긴 필드 저장 구조를 바탕으로 한 것이다. 즉, EXODUS에서와 동일한 B* 트리 인덱스를 사용하며 Starburst에서처럼 버디 시스템을 사용하여 B* 트리의 일 노드인 세그먼트를 할당 및 반환한다. 그런데 세그먼트는 EXODUS에서처럼 일정 크기로 고정되어 있거나 Starburst에서처럼 두배씩 크기가 증가하도록 제약되어 있지 않고, 임의의 크기로 할당될 수 있도록 하였다.

2.2 BLOB 타입의 성능 평가

BLOB 타입의 성능 평가 연구는 BLOB 타입의 효율적인 저장 구조를 제시하려는 연구에 비하여 많이 수행되지 않았다. [6]에서는 EXODUS의 큰 저장 객체를 저장하기 위한 B* 트리에 기반을 둔 저장 구조를 제시하면서 그 성능 평가를 수행하였다. EXODUS의 큰 저장 객체 구조의 성능은 검색, 삽입, 삭제, 수정, 첨부(append) 등의 연산, 특히 일정 바이트 범위의 데이터에 대한 부분 연산에 소요되는 I/O 비용 및 데이터 변경에 따른 저장 공간 효율을 측정함으로써 평가하였다. 이 성능 평가에서는 CLOCK 알고리즘을 사용하는 버퍼 관리자 및 큰 저장 객체가 저장되는 디스크 공간을 시물레이션하여 실제 디스크 페이지에 대한 I/O를 수행하는 대신 I/O의 횟수를 셈으로써 I/O 시간을 측정하였다. 이 시물레이션에 사용된 주요 파라미터는 표 1과 같다. 표 1에서 연산의 크기란 부분 연산의 대상이 되는 바이트 범위의 크기를 의미한다. 연산 배분(operation mix)은 삽입과 삭제의 양을 동일하게 함으로써 큰 저장 객체의 크기를 일정하게 유지하였다. 디스크 접근 시간은 무작위 접근(random access)에 대한 것으로 탐색 시간(seek time)과 회전지연 시간(rotational latency)을 합한 것이다.

EXODUS에서 큰 저장 객체의 성능 평가는 타 시스템의 큰 객체 또는, 긴 필드 저장 구조들과의 상대적인 성능 비교가 아니고 EXODUS의 그것에 대한 절대 평가에 그쳤다. 이에 반하여 [8]에서는 EXODUS 및 Starburst의 저장 구조들과 이들을 개선한 EOS의 저장 구조 3가지를 상대 평가 비교하였다. 이 연구에서도 [6]

의 :
공간
객체
크기
셈으
다,
들을
EOS:
객체
분 검
공간

표 1

| |
|------|
| 디스크 |
| A |
| 큰 객체 |
| |
| |
| |
| 디스크 |
| 데이터 |

3. 1

본 절
일 BLC
고 처리
특성을
타미더
을 제시
해야 하
임의 비
타입 다
3.1 E
BLOB
타입으로
타에 대한
어 데이터
구조 및

의 평가에서와 마찬가지로 큰 객체가 저장되는 디스크 공간은 실제로 구현하지 않고 시뮬레이션하였다. 즉, 큰 객체에 대한 접근은 실제로 수행되는 것이 아니라 디스크 탐색 및 세그먼트를 구성하는 페이지 전송의 횟수를 셈으로써 큰 객체에 대한 연산의 I/O 비용을 측정하였다. 시뮬레이션 파라미터도 [6]의 연구에서와 동일한 것들을 사용하였다. 성능은 EXODUS, Starburst, 그리고 EOS가 제공하는 큰 객체 또는 긴 필드의 저장 구조를 객체 생성 시간, 객체 전체에 대한 순차 검색 시간, 부분 검색 및 변경 시간, 그리고 부분 변경에 따른 저장 공간 효율 등을 측정함으로써 평가하였다.

표 1 기존 연구에서 사용된 BLOB 타입 성능 평가 파라미터

| 파라미터 | 값 |
|-------------|------------------------|
| 디스크 페이지 크기 | 4K 바이트 |
| 세그먼트 크기 | 1 페이지 및 4 페이지 |
| 큰 저장 객체의 크기 | 10M 바이트 및 100M 바이트 |
| 연산의 크기 | 100 바이트 및 10K 바이트 |
| 연산 배분 | 40% 검색, 30% 삽입, 30% 삭제 |
| 버퍼 용량 | 12 페이지 |
| 디스크 접근 시간 | 33ms |
| 데이터 전송 시간 | 1K 바이트/ms |

3. 다양한 BLOB 타입

본 절에서는, 기존 연구들에서처럼 범용성을 갖는 단일 BLOB 타입으로 모든 멀티미디어 데이터를 저장하고 처리하는 것보다, 멀티미디어 데이터베이스 응용들의 특성을 반영하는 특성화된 다양한 BLOB 타입들로 멀티미디어 데이터를 저장하고 처리하는 것이 더 효율적임을 제시한다. 먼저 3.1절에서 단일 BLOB 타입이 만족해야 하는 요건을 살펴보고, 3.2절에서 단일 BLOB 타입의 비효율성을 지적하고, 3.3절에서 그에 따른 BLOB 타입 다양화의 필요성을 제시한다.

3.1 단일 BLOB 타입의 요건

BLOB 타입이란 DBMS가 제공하는 하나의 데이터 타입으로서 데이터를 저장하는 저장 구조와 저장된 데이터에 대한 연산으로 구성된다. 따라서, 임의의 멀티미디어 데이터를 저장하기 위한 단일 BLOB 타입은 저장 구조 및 연산의 범용성을 확보하여야 한다. 기존 연구들

에서 이와 같은 단일 BLOB 타입을 실현하기 위하여 가장 중점을 두었던 점은 다음 2가지로 요약할 수 있다:

1. 저장되는 BLOB의 크기에 제약이 없어야 한다. 즉, 물리적인 저장 장치의 공간 범위 내에서 크기가 무제한인 BLOB을 저장할 수 있어야 한다.
2. 일정 바이트 범위에 대한 부분 연산 특히, 부분 변경 연산을 효율적으로 처리할 수 있어야 한다.

전자는 멀티미디어 데이터가 일반적으로 대용량이라는 점에서 타당성을 가지며, 후자는 바로 그러한 길고 큰 데이터에 대해서는 데이터 전체에 대한 연산보다는 일정 바이트 범위에 대한 부분 연산이 더 많이 요구될 것이라는 점에서 타당성을 갖는다.

[6]에서 제시된 EXODUS의 큰 저장 객체는 바로 이러한 두 요건을 만족하는 전형적인 단일 BLOB 타입이다. EXODUS의 B⁺ 트리 인덱스는 다단계 인덱스로서 인덱스 자체에 대한 크기 제한이 없으므로, 무제한의 크기를 갖는 멀티미디어 데이터를 여러 세그먼트에 나누어 저장할 수 있도록 지원하고 있다. 또한 이 B⁺ 트리는 멀티미디어 데이터의 임의의 바이트 위치를 직접 접근하는 데 이용되며 무제한의 크기가 허용되는 멀티미디어 데이터에 대한 부분 연산을 수행할 때 없어서는 안될 부분이다.

한편, 실제 멀티미디어 데이터를 세그먼트에 저장하는 데 있어 하나 또는 소수의 큰 세그먼트에 저장하지 않고 일반적으로 멀티미디어 데이터의 큰 용량에 비하여 상대적으로 적은 크기의 여러 세그먼트들에 나누어 저장하는 이유는, 부분 삽입, 삭제, 그리고 수정과 같은 부분 변경 연산을 효율적으로 지원하기 위해서이다. 그 이유를 설명하면 다음과 같다:

멀티미디어 데이터는 일반적으로 대용량이기 때문에 변경이 일어날 경우 고장 회복을 위하여 새도우 페이징(shadow paging)되는 것이 보통이다 [10]. 이는 일반적으로 DBMS에서 널리 쓰이는 로깅이 멀티미디어 데이터에 대해서는 너무 큰 부담을 낳기 때문이다. 그런데 멀티미디어 데이터를 가급적 물리적으로 인접한 디스크 공간에 저장(clustering)하기 위해서 사용되는 세그먼트는 바로 물리적으로 인접한 디스크 페이지들의 모임이기 때문에, 한 세그먼트 내의 부분 변경을 수행하기 위해서는 해당 세그먼트 전체를 새도우잉시킨 후 디스크 공간 내의 다른 위치에 새로운 세그먼트를 할당받아 변경을 처리해야 한다. 이러한 변경 과정은 세그먼트의 크기가 크고 따라서 상대적으로 부분 변경되는 바이트 범위가

작을 경우, 너무 큰 부담이 된다. 뿐만 아니라 부분 연산의 비용이 대상 바이트 범위의 크기에 의해 결정되는 것이 아니라 해당 세그먼트의 전체 크기에 좌우되는 현상을 낳게 된다. 따라서 부분 변경 연산을 효율적으로 처리하기 위해서는 세그먼트 크기에 적절한 제한이 있어야 한다.

3.2 단일 BLOB 타입의 비효율성

본절에서는, 단일 BLOB 타입이 만족해야 하는 주요 요건인 3.1절에서 언급한 데이터 크기 무제한 요건과 부분 연산 지원 요건의 문제점을 검토한다.

3.2.1 무제한 크기 요건

일반적으로 멀티미디어 데이터는 대용량으로 Giga 바이트 규모 등의 큰 데이터가 효율적으로 저장될 수 있어야 하는 것은 멀티미디어 DBMS에서는 당위라 하겠다. 그러나 여러 응용들이 필요로 하는 멀티미디어 데이터들이 모두 무제한의 크기를 요하는 것은 아니다.

예를 들어 보자. 표 2는 사운드 카드가 설치된 PC에서 보이스 에디터로 3가지 서로 다른 샘플링 비율로써 사운드를 각각 10초, 20초, 30초 씩 녹음했을 때 만들어진 사운드 데이터의 크기를 나타낸 것이다. 장시간에 걸친 연설이라든지 음악 연주의 경우 해당 데이터의 크기는 압축을 하더라도 거의 무제한의 크기에 해당하는 대용량일 것이다. 그러나 표 2에서 보듯이 멀티미디어 프리젠테이션에서 짧은 음성 주(annotation)라든지 음성 안내 시스템의 간단한 음성 서비스와 같은 응용의 경우라면 비교적 크기 작은 음성 데이터를 사용하게 될 것이다.

표 2 사운드 데이터의 크기

| 녹음 시간 | 샘플링 비율 | | |
|-------|----------|----------|----------|
| | 8,070Hz | 11,900Hz | 15,000Hz |
| 10초 | 65K 바이트 | 114K 바이트 | 142K 바이트 |
| 20초 | 115K 바이트 | 230K 바이트 | 300K 바이트 |
| 30초 | 229K 바이트 | 261K 바이트 | 442K 바이트 |

표 3은 binary level, gray level, 그리고 color level 픽셀(pixel) 당 각각 1 bit, 1 바이트, 3 바이트를 사용할 경우 이미지 데이터의 크기를 나타낸 것이다. 이를 보면, 얼굴 증명사진이라든지 간단한 약도를 다루는 응용의 경우 이미지 데이터의 크기가 무제한적이지 않음을 알 수 있다.

표 3 이미지 데이터의 크기

| | 그림 크기(pixel 수) | 크기 |
|--------------|----------------|----------|
| binary level | 256 × 256 | 8K 바이트 |
| | 1024 × 768 | 96K 바이트 |
| gray level | 256 × 256 | 64K 바이트 |
| | 1024 × 768 | 768K 바이트 |
| color level | 256 × 256 | 192K 바이트 |
| | 1024 × 768 | 2.3M 바이트 |

대용량 멀티미디어 데이터를 저장하기 위해서는, EXODUS의 B⁺ 트리와 같은 인덱스가 필요하다. 이는 대용량 멀티미디어 데이터가 디스크 공간 내의 여러 세그먼트들에 나뉘어 저장될 경우 세그먼트 디렉토리의 역할을 할 뿐 아니라, 대용량 데이터에 대한 부분 연산시 임의의 바이트 위치로 바로 접근하기 위한 것이다. 이러한 인덱스가 제공되지 않는다면 대용량 데이터에 대한 부분 연산의 비용이 연산 대상 데이터의 크기 대신에 데이터 전체 크기에 좌우됨으로써 결과적으로 부분 연산을 효율적으로 지원하지 못하게 된다.

그러나 이러한 인덱스 구조는 표 2나 표 3에서처럼 멀티미디어 데이터의 크기가 적절히 제한되는 응용에 대해서는 필요없는 부담이 될 뿐이다. 심지어 대용량의 데이터라 할지라도, 부분 연산보다는 예를 들어 음악 연주 녹음 데이터의 플레이와 같이, 해당 응용에서 데이터의 처음부터 마지막까지 순차적으로 전체 검색만을 필요로 하는 경우라면 이러한 인덱스는 불필요하다.

3.2.2 부분 연산 지원 요건

부분 연산 지원 요건도 응용의 특성과 멀티미디어 데이터의 처리 요건에 따라 전적으로 또는 부분적으로 불필요할 수 있다. 부분 연산에는 주어진 바이트 범위에 대한 검색, 삽입, 삭제, 수정, 그리고 데이터의 맨 뒤에 새 데이터를 삽입하는 첨부가 있다. 단일 BLOB 타입은 이들 모든 부분 연산을 효율적으로 지원하여야 한다. 그러나 모든 응용들이 멀티미디어 데이터를 부분 접근하는 것을 요구하지 않는다.

선곡된 음악을 대중을 상대로 플레이하는 것이라든지, 전자 정보 안내소(kiosk)에서 사용자의 버튼 조작에 따라 어떤 특정 안내 자료 일체를 일방적으로 디스플레이하는 것이라든지, VOD 서버로부터 영화 한편을 가입자의 요청에 따라 상영하는 경우처럼 데이터의 처음부터 끝까지 전체를 순차적으로 검색하는 응용에서는 임의의

바이
하는
부
다. 9
타에
경우
된 사
다. 비
타의
교체
내기
의 텍
분 변
작 과
완성
텍스트
보 서
타들은
부분
수한
다. 정
를 생
로운
에 부
스를
보 이
부분
를 추
서비스
있다.
수정
들이다.
3.3
단일
입의
위한
더 효
예를
부분
는 멀
읽기
첨부
적합
단일
멀티

바이트 위치 및 범위가 주어져 해당 데이터를 부분 검색하는 것이 필요 없다.

부분 변경이 필요 없는 멀티미디어 응용도 많이 있다. 얼굴 증명 사진의 예를 들어 보자. 이 이미지 데이터에 대한 부분 변경 연산은 몽타췌 제작과 같은 응용의 경우에는 필요하겠지만 인사 관리 데이터베이스에 저장된 사원들의 얼굴 사진에 대해서는 전혀 불필요할 것이다. 비디오, 오디오, 텍스트 등의 데이터는, 비디오 데이터의 제작 과정에서 일부 비디오 프레임을 새 내용으로 교체한다든지, 오디오 녹음 과정에서 특수 음향 효과를 내기 위하여 해당 부분을 여러차례 수정한다든지, 장문의 텍스트를 계속 부분 부분 첨삭 및 수정하기 위한 부분 변경을 필요로 한다. 그러나 이러한 부분 변경은 제작 과정에서 필요한 것이지 최종적으로 데이터의 제작이 완성되어 읽기 전용(read only)의 비디오, 오디오, 또는 텍스트로 제공되면서부터는 불필요하다. 일반적으로 정보 서비스 및 검색 시스템이 제공하는 멀티미디어 데이터들은 대부분 읽기 전용이다.

부분 변경이 필요하지 모든 형태의 연산이 아니라 특수한 형태의 연산만 요구되는 멀티미디어 데이터도 있다. 정보 서비스 시스템이 제공하는 뉴스 텍스트 데이터를 생각해 보자. 이 데이터는 현재의 내용에 시시각각 새로운 뉴스를 추가함으로써 변경된다. 즉, 기존의 데이터에 부분 삽입을 수행하여 데이터를 부분 변경하는데 뉴스를 보도 시간순으로 정리하여 제공하는 서비스의 경우 이 부분 삽입은 바로 기존 데이터의 맨 뒤에 새 데이터를 추가하는 이른바 첨부 연산이 된다. 일반적으로 정보 서비스 시스템에는 이러한 유형의 텍스트 데이터가 많이 있다. 즉, 변경 연산으로는 일반적인 부분 삽입, 삭제, 수정은 필요로 하지 않고 오직 첨부만을 필요로 하는 것들이다.

3.3 BLOB 타입 다양화의 필요성

단일 BLOB 타입의 비효율성은 곧, 단일 BLOB 타입의 범용성을 필요로 하지 않는 멀티미디어 데이터를 위한 특성화된 BLOB 타입들이 단일 BLOB 타입보다 더 효율적일 수 있음을 의미한다.

예를 들어, 크기에 제한이 있는 멀티미디어 데이터, 부분 검색은 필요로 하지 않고 전체 검색만을 필요로 하는 멀티미디어 데이터, 부분 변경은 필요로 하지 않는 읽기 전용의 멀티미디어 데이터, 그리고 부분 변경으로 첨부 연산만을 필요로 하는 멀티미디어 데이터 각각에 적합한 BLOB 타입이 해당 데이터를 저장하는 데 있어 단일 BLOB 타입보다 더 효율적일 것이다.

멀티미디어 데이터의 크기가 어느 정도이며 그에 따

라 효율적인 저장 구조가 어떤 것인가 또는 멀티미디어 데이터에 대한 전체 검색, 부분 검색, 부분 변경 등의 연산 중 어느 것이 가장 자주 수행되고 따라서 가장 효율적으로 지원되어야 하는 것은, 전적으로 응용의 특성과 데이터 처리 요건에 달려 있다. 중요한 것은 현재까지 알려진 여러 멀티미디어 응용들은 3.2절에서 예를 든 것처럼 대부분 멀티미디어 데이터에 대하여 특정 연산 위주의 처리를 필요로 한다는 점이다.

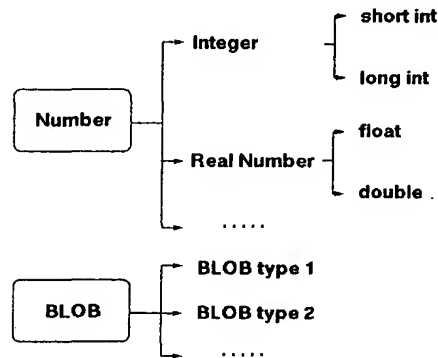


그림 3 BLOB 타입 다양화의 필요성

따라서 단일 BLOB 타입에 만족할 것이 아니라, BLOB 타입 다양화의 필요성이 있다. 그리하여 멀티미디어 데이터를 모델링하거나 저장할 때 사용자로 하여금 해당 데이터의 특성과 처리 요건에 가장 적합한 BLOB 타입을 선택하도록 하는 것이 모든 멀티미디어 데이터를 단일 BLOB 타입으로 모델링하고 저장하는 것에 비해 더 효율적이다. 이는 프로그래밍 언어에서 수 데이터 타입의 구현에 비유하여 설명할 수 있다. 만약, 정수형, 실수형 등의 다양한 수 데이터 타입을 단일 타입으로써 구현한다면 아주 비효율적일 것이다. 수 데이터의 여러 특성에 적합한 다양한 타입들 (예를 들어 C 언어의 경우, short, int, long, float, double 등)을 제공함으로써 수 데이터를 저장하듯이 BLOB에 대해서도 다양한 타입들이 제공되는 것이 효율적이다 (그림 3 참조).

4. 성능 평가

본 절에서는 본 논문에서 제시하는 BLOB 타입 다양화의 효율성을 평가한 결과를 기술한다. 이 성능 평가는, 멀티미디어 DBMS가 단일 BLOB 타입을 제공하여 모든 BLOB을 이에 저장하는 경우와 여러 다양한 BLOB 타입들을 제공하여 BLOB 별로 응용 상의 특성

과 처리 요건에 적합한 BLOB 타입을 선택하여 저장하는 경우를 비교하였을 때, 후자가 전자에 비하여 얼마나 더 효율적인지를 알아보기 위한 것이다.

이 평가는 3가지의 특성화된 BLOB 타입들을 기존 관계 DBMS 프로토타입의 저장 시스템 상에 구현한 후 실험을 통하여 수행하였다. 먼저 4.1절에서 본 BLOB 타입 성능 평가의 개요를 설명하고, 4.2절에서 BLOB 타입의 구현을 기술하고, 4.3절에서 성능 평가 실험 및 그 결과를 기술한다.

4.1 개요

성능 평가는 범용성을 갖지 않지만 실제 멀티미디어 데이터베이스 응용에서 많이 사용되는 2가지 BLOB 타입과 범용성을 갖는 단일 BLOB 타입을 대상으로 수행하였다. 첫째 타입은 부분 변경 연산을 지원하지 않는 읽기 전용 BLOB 타입이고, 둘째 타입은 검색 연산과 더불어 부분 변경으로는 첨부 연산만을 지원하는 첨부 전용 BLOB 타입이고, 셋째 타입은 기존 연구들이 제시한 단일 BLOB 타입이다. 본 논문에서는 이하 이들 세 타입을 각각 R-BLOB(Read Only BLOB 타입), A-BLOB(Append Only BLOB 타입), 그리고 G-BLOB(General-Purpose BLOB 타입)이라 부른다.

성능 평가 방법은, (1) 읽기 전용 BLOB을 R-BLOB에 저장하는 경우와 G-BLOB에 저장하는 경우에 있어 전체 검색과 부분 검색의 I/O 비용을 비교하는 것과 (2) 첨부 전용 BLOB을 A-BLOB에 저장하는 경우와 G-BLOB에 저장하는 경우에 있어 첨부, 전체 검색, 그리고 부분 검색의 I/O 비용을 비교하는 것이다.

I/O 비용은 기존의 BLOB 성능 평가 연구 [6][8]에서와 같이 정의하였다. 즉, 검색 연산의 경우에는 BLOB 공간의 세그먼트에 대한 디스크 탐색 시간과 세그먼트를 구성하는 페이지 전송 시간의 합이고, 변경 연산의 경우에는 변경과 고장 회복을 위한 페이지 새도우에 소요되는 디스크 탐색 시간과 페이지 전송 시간의 합이다.

이들 비교를 통하여, 읽기 전용 BLOB은 G-BLOB 보다는 R-BLOB에 저장하는 것이, 그리고 첨부 전용 BLOB은 G-BLOB보다는 A-BLOB에 저장하는 것이 얼마나 더 효율적인가를 평가하였다. 즉, 단일 BLOB 타입에 모든 멀티미디어 데이터를 저장하는 것보다 다양한 BLOB 타입을 제공하여 각 BLOB의 특성에 적합한 타입을 선택하여 저장하는 것이 얼마나 더 효율적인가를 알아 보았다.

성능 평가는 이미 개발된 PC용 관계 DBMS 프로토타입의 저장 시스템 상에 R-BLOB, A-BLOB, 그리고

G-BLOB을 각각 구현한 후, 실험을 통하여 수행하였다. BLOB 타입의 구현, 실험 방법, 그리고 실험 파라미터들은 모두 기존의 BLOB 타입 성능 평가 연구 [6][8]에서와 동일하게 하였고, 실험은 상기 관계 DBMS가 운용되는 DOS 환경의 486 PC 상에서 수행하였다.

4.2 BLOB 타입의 구현

본 절에서는 성능 평가에 사용된 BLOB 타입의 구현을 기술한다. 4.2.1절에서는 먼저 구현에 사용된 관계 DBMS에 대하여 소개하고, 4.2.2절에서는 R-BLOB, A-BLOB, 그리고 G-BLOB의 구현을 기술한다.

4.2.1 구현에 사용된 관계 DBMS 개요

본 관계 DBMS 프로토타입은 DOS 환경의 PC용 DBMS로서 WiSS의 구조와 유사한 여러 층(layer)의 구조를 갖는 시스템이다. 사용자 인터페이스를 겸한 Parser, CE(command executor), AM(access method), SS(storage structure), 그리고 BM(buffer manager)의 5층이 DOS 화일 시스템 상에 구현되어 있다. Parser와 CE는 질의 처리기를 구성하고, AM, SS, BM의 3층은 DOS 화일 시스템 상에서 저장 시스템을 구성한다. 각 층은 바로 아래 층이 제공하는 인터페이스를 통해 아래 층의 기능을 제공받아 자신 층의 기능을 구현하고 바로 위 층에게 자신의 인터페이스를 통해 자신의 기능을 제공한다. 그림 4는 본 관계 DBMS의 층 구조 및 층 간의 인터페이스를 나타낸 것이다.

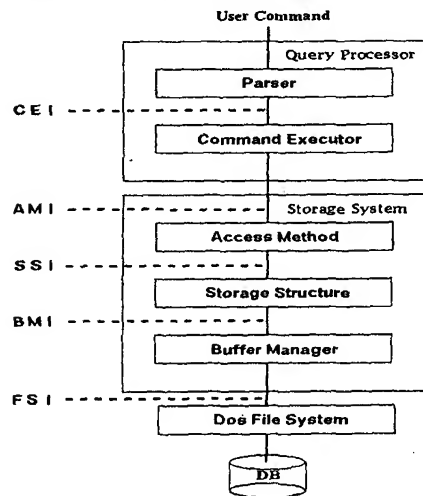


그림 4 구현에 사용된 관계 DBMS의 층 구조

그림 4에서 CEI, AMI, SSI, BMI, FSI는 각각 CE, AM, SS, BM, DOS 화일 시스템이 자신의 위 층에 제

공하
면 디
제공
구문
행 루
는 층
설정
사용지
로서,
등 튜
층으로
기능을
막으로
연산을
수행한
4.2.
본
G-BLOC
I/O 비
변경을
이들의
것들인
기에 제
BLOB
목적이
BLOB
데이터
곳 즉,
저장하
능 평가
시물레이
을 측정
스크 탐색
을 측정
각각에
데이터베
BLOB
공간, 그
공간을 상
구현된
Starburst
조를 갖
플의 해
criptor)

공하는 인터페이스를 나타낸다. 각 층의 기능을 설명하면 다음과 같다. Parser는 간단한 사용자 인터페이스를 제공하여 사용자의 명령어 (관계 질의어)를 입력 받아 구문 분석을 수행하고 CEI가 제공하는 명령 별 해당 수행 루틴을 호출한다. CE는 사용자의 명령을 수행해 주는 층으로서 AMI가 제공하는 릴레이션의 개폐, scan의 설정, 튜플의 삽입, 삭제, 검색 등의 기능을 이용하여 사용자 명령을 수행한다. AM은 AMI를 구현하는 층으로서 SSI가 제공하는 화일의 개폐, scan의 내부 설정 등 튜플 단위의 접근을 수행한다. SS는 SSI를 구현하는 층으로서 BMI가 제공하는 디스크 페이지의 버퍼 관리 기능을 이용하여 페이지 단위의 접근을 수행한다. 마지막으로 BM은 DOS 화일 시스템이 제공하는 화일 조작 연산을 이용하여 디스크 입출력 및 LRU 버퍼 관리를 수행한다.

4.2.2 R-BLOB, A-BLOB, G-BLOB의 구현

본 성능 평가를 위하여 구현한 R-BLOB, A-BLOB, G-BLOB의 저장 구조는 R-BLOB의 경우에는 읽기 I/O 비용이 최소가 되도록, A 및 G-BLOB의 경우에는 변경을 위한 I/O 비용을 줄일 수 있도록 설계하였다. 이들의 저장 구조는 모두 BLOB 크기에 제한을 두는 것들인데 물론 R-BLOB, A-BLOB, G-BLOB 각각 크기에 제한이 없는 것이 있었으나 본 구현에서는 다양한 BLOB 타입들과 단일 BLOB 타입 간의 성능 비교가 목적이므로 간단한 구현을 위하여 크기에 제한이 있는 BLOB 타입으로 구현하였다.

데이터베이스 공간(DB area)을 BLOB을 저장하는 곳 즉, BLOB 공간(BLOB space)과 그외의 데이터를 저장하는 곳으로 나누었고, BLOB 공간은 [6][8]의 성능 평가 연구에서와 마찬가지로 실제로 구현하지 않고 시뮬레이션하였다. 즉, BLOB에 대한 연산의 I/O 비용을 측정함에 있어 디스크 접근을 직접 수행하지 않고 디스크 탐색 및 페이지 전송의 횟수를 셈으로써 I/O 비용을 측정하였다. BLOB 공간은 상기의 세 BLOB 타입 각각에 대하여 별도의 BLOB 공간을 할당하였다. 즉, 데이터베이스 내의 모든 R-BLOB들이 저장되는 R-BLOB 공간, 모든 A-BLOB들이 저장되는 A-BLOB 공간, 그리고 모든 G-BLOB들이 저장되는 G-BLOB 공간을 상정하였다.

구현된 이들 BLOB 타입들은 모두 EXODUS, Starburst, EOS에서와 같이 세그먼트 기반의 저장 구조를 갖도록 설계하였고, BLOB의 저장은 릴레이션 튜플의 해당 BLOB 필드에 BLOB 서술자(BLOB descriptor)를 저장하고, 실제 BLOB은 해당 BLOB 공간

의 세그먼트(들)에 저장함으로써 이루어지도록 하였다.

R-BLOB의 저장 구조는 R-BLOB 공간 내의 단일 세그먼트에 BLOB 전부를 저장하는 것이다³⁾. 이때, 해당 튜플 내 BLOB 필드의 BLOB 서술자는 다음 두 값으로 구성된다 (그림 5 참조).

1. R-BLOB 공간 내의 세그먼트 식별자(segment identifier)
2. 세그먼트의 길이(segment length)

R-BLOB에서는 전체 검색 및 부분 검색 연산만이 제공되므로 상기와 같이 단일 세그먼트에 BLOB을 모두 저장하면, 검색 연산은 1회의 디스크 탐색과 순차적인 페이지 전송으로 처리된다. 따라서 I/O 비용은 최적이며 저장 공간 효율도 100%이다.

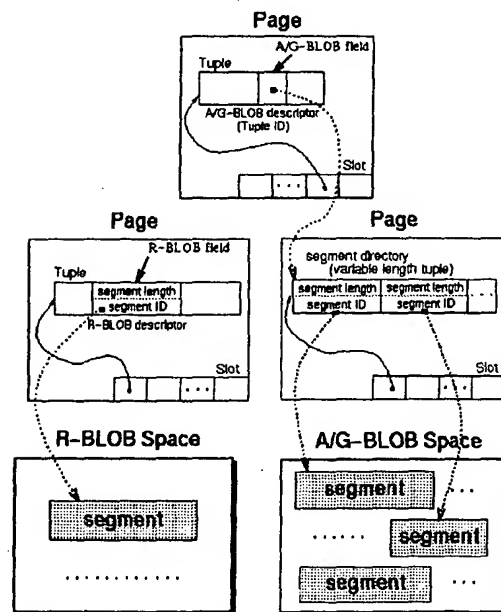


그림 5 R/A/G-BLOB의 저장 구조

A-BLOB의 저장 구조는 A-BLOB 공간 내에 할당된 가변 길이의 세그먼트들에 BLOB을 나누어 저장하고, 부분 검색을 위하여 이들 세그먼트들에 대한 디렉토리를 제공하는 것이다. 이때 할당되는 세그먼트들의 길이는

3) EOS의 경우, 디스크 페이지의 크기가 4K 바이트라고 할 때 최대 635M 바이트 크기의 버퍼 공간 내에서 지원할 수 있는 단일 세그먼트의 최대 크기는 32M 바이트이다 [3].

첨부 연산이 첨부하는 데이터의 크기에 의해 결정된다. 따라서 큰 양의 데이터가 첨부되면 큰 크기의 세그먼트가 할당되고 적은 양의 데이터가 첨부되면 작은 크기의 세그먼트가 할당된다. 단, 너무 작은 크기의 세그먼트가 많이 할당되어 전체 또는 부분 검색시 잦은 디스크 탐색 부담을 초래하지 않도록 하기 위하여, 할당되는 세그먼트의 최소 크기를 16K 바이트로 정하였다. 이는 4K 바이트 페이지 4개가 모인 것으로 EXODUS에서 제시한 세그먼트의 크기와 같은 것이다.

부분 검색을 위한 세그먼트 디렉토리는 (A-BLOB 공간 내의 세그먼트 식별자, 세그먼트 길이) 쌍들의 집합이다. 본 구현에서는 이 세그먼트 디렉토리를 가변 길이 튜플로 구현하여 일반 릴레이선의 튜플처럼 저장되도록 하였다. 따라서 BLOB을 저장하는 튜플 내 BLOB 필드의 BLOB 서술자는 이 세그먼트 디렉토리를 가리키는 튜플 식별자(tuple identifier)로 구현하였다. 본 구현에 사용된 관계 DBMS의 저장 시스템에서는 튜플을 저장하기 위하여 System R의 슬롯(slot) 기반 페이지 구조를 사용하고 있다. 따라서 BLOB 서술자로서 저장된 튜플 식별자는 세그먼트 디렉토리가 튜플로 저장되어 있는 페이지의 해당 슬롯을 가리키는 포인터이다 (그림 5 참조).

G-BLOB의 구현은, 그 저장 구조가 EXODUS에서처럼 G-BLOB 공간 내에 할당된 세그먼트들이 모두 4K 바이트 페이지 4개로 구성된 고정 길이 (즉, 16K 바이트 세그먼트)로 국한된다는 점을 제외하고는, A-BLOB의 구현과 동일하다. 세그먼트의 크기를 16K 바이트로 국한한 이유는 EXODUS에서처럼 부분 변경 연산 때 페이지 새도우잉의 부담을 적정 수준으로 유지하기 위해서이다.

4.3 성능 평가 실험 및 결과

표 4 성능 평가 항목 및 실험 결과

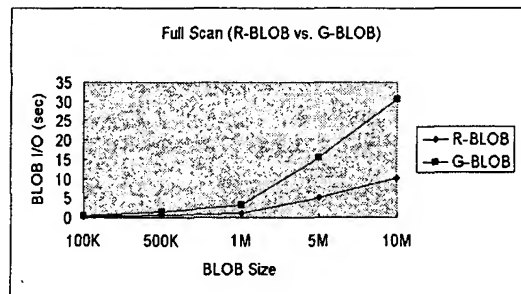
| 평가 항목 | 실험 결과 |
|--|-------|
| 읽기 전용 BLOB을 R-BLOB과 G-BLOB에 각각 저장하는 경우, 전체 검색(full scan) I/O 비용 | 그림 6 |
| 읽기 전용 BLOB을 R-BLOB과 G-BLOB에 각각 저장하는 경우, 부분 검색(partial scan) I/O 비용 | 그림 7 |
| 첨부 전용 BLOB을 A-BLOB과 G-BLOB에 각각 저장하는 경우, 첨부(append) I/O 비용 | 그림 8 |
| 첨부 전용 BLOB을 A-BLOB과 G-BLOB에 각각 저장하는 경우, 전체 검색 I/O 비용 | 그림 9 |
| 첨부 전용 BLOB을 A-BLOB과 G-BLOB에 각각 저장하는 경우, 부분 검색 I/O 비용 | 그림 10 |

표 4는 성능 평가 항목 및 실험 결과 일람표이다. 그

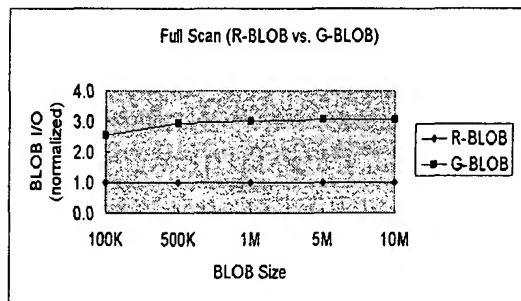
림 6에서 그림 10은 각각 해당 I/O 비용을 (a) 시간(sec)으로 측정하여 비교한 것과 (b) 정규화(normalization)하여 비교한 것으로 구성되어 있다. 이들 평가에 사용된 파라미터들은 표 5에 정리하였는데, 기존의 BLOB 성능 평가 연구 [6][8]에서와 같은 값들이다. 각 성능 평가 항목에 대하여 그 실험 방법 및 결과를 기술하면 다음과 같다.

표 5 성능 평가 파라미터

| 파라미터 | 값 |
|-----------------------|-----------------------------|
| 디스크 페이지 크기 | 4K 바이트 |
| G-BLOB의 고정 길이 세그먼트 크기 | 4 페이지 |
| BLOB의 크기 | 100K, 500K, 1M, 5M, 10M 바이트 |
| 연산의 크기 | 10K, 100K, 500K, 1M 바이트 |
| 디스크 탐색 시간 | 33ms |
| 데이터 전송 시간 | 1K 바이트/ms |



(a)



(b)

그림 6 R-BLOB과 G-BLOB의 전체 검색 I/O 비용 비교

R-BLOB과 G-BLOB의 전체 검색 I/O 비용은, BLOB의 크기를 100K, 500K, 1M, 5M, 10M 바이트 등으로 변화시키며 각각에 대하여 측정하였다. BLOB의

크기가
전체 검색
전체
R-BLOC
여 완
BLOB
G-BLO
됨을 보
R-BI
10M 바
크기 (즉
500K, 1
정하였다
바이트
I/O 비
가 비교
검색 비
검색 비
의 경우
을 보이
R-BLOB
하면 부
약 3배에

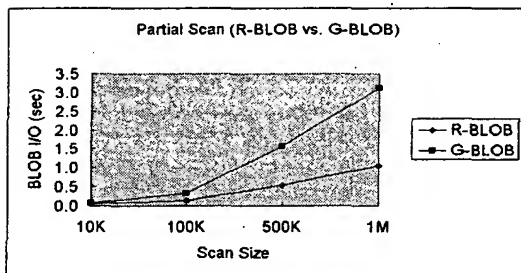
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10

3.0
2.0
1.0
0.0
10

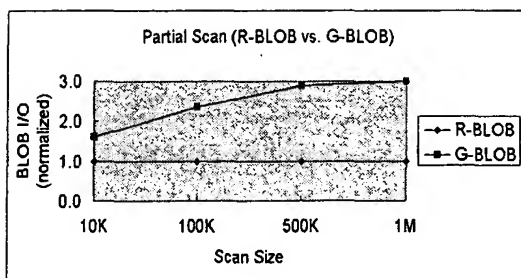
그림 7)

크기가 비교적 작을 때는 R-BLOB과 G-BLOB 모두 전체 검색 비용이 낮지만 BLOB의 크기가 점차 커질수록 전체 검색 비용이 증가함을 나타내고 있다. 그런데 R-BLOB의 경우 그 증가폭은 G-BLOB의 그것에 비하여 완만함을 보이고 있다 (그림 6(a)). 읽기 전용 BLOB을 R-BLOB에 저장하는 경우에 비하여 G-BLOB에 저장하면 전체 검색 비용이 약 3배 정도가 됨을 보여준다 (그림 6(b)).

R-BLOB과 G-BLOB의 부분 검색 I/O 비용은, 10M 바이트 크기의 읽기 전용 BLOB에 대하여 검색의 크기 (즉, 검색되는 바이트 범위의 크기)를 10K, 100K, 500K, 1M 바이트 등으로 변화시키며 각각에 대하여 측정하였다. 각 검색의 크기에 대하여 무작위로 100개의 바이트 범위를 생성하여 해당 데이터를 검색하고 그 I/O 비용을 측정하여 평균값을 구하였다. 검색의 크기가 비교적 작을 때는 R-BLOB과 G-BLOB 모두 부분 검색 비용이 낮지만 검색의 크기가 점차 커질수록 부분 검색 비용이 증가함을 나타내고 있다. 그런데 R-BLOB의 경우 그 증가폭은 G-BLOB의 그것에 비하여 완만함을 보이고 있다 (그림 7(a)). 읽기 전용 BLOB을 R-BLOB에 저장하는 경우에 비하여 G-BLOB에 저장하면 부분 검색 비용이 검색의 크기가 증가해 감에 따라 약 3배에 가까워짐을 보여준다 (그림 7(b)).



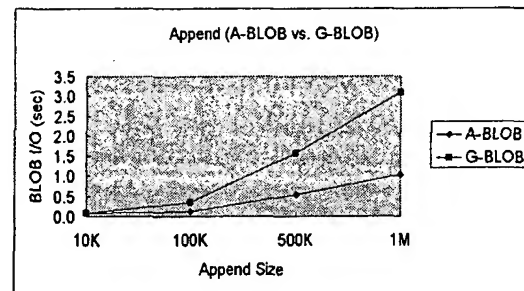
(a)



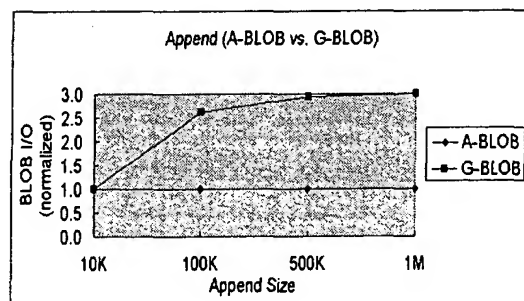
(b)

그림 7 R-BLOB과 G-BLOB의 부분 검색 I/O 비용 비교

A-BLOB과 G-BLOB의 첨부 연산 I/O 비용은, 10M 바이트 크기의 첨부 전용 BLOB에 대하여 첨부부의 크기 (즉, 한번에 첨부되는 데이터의 크기)를 10K, 100K, 500K, 1M 바이트 등으로 변화시키며 각각에 대하여 측정하였다. 다시 말해서 BLOB의 크기가 10M 바이트가 될 때까지 한번에 10K 바이트 (또는 100K, 500K, 1M 바이트) 씩 첨부해 나갈 경우 각각의 첨부 연산의 평균 I/O 비용을 구하였다. 첨부부의 크기가 비교적 작을 때는 A-BLOB과 G-BLOB 모두 첨부 비용이 낮지만 첨부부의 크기가 점차 커질수록 첨부 비용이 증가함을 나타내고 있다. (특히, 첨부부의 크기가 A-BLOB의 최소 세그먼트 크기인 16K 바이트보다 작을 때는 A-BLOB과 G-BLOB의 첨부 비용이 동일함을 보이고 있다.) 그런데 A-BLOB의 경우 그 증가폭은 G-BLOB의 그것에 비하여 완만함을 보이고 있다 (그림 8(a)). 첨부 전용 BLOB을 A-BLOB에 저장하는 경우에 비하여 G-BLOB에 저장하면 첨부 비용이 첨부부의 크기가 증가해 감에 따라 약 3배에 가까워짐을 보여준다 (그림 8(b)).



(a)

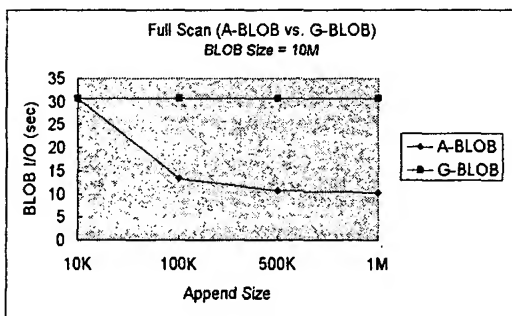


(b)

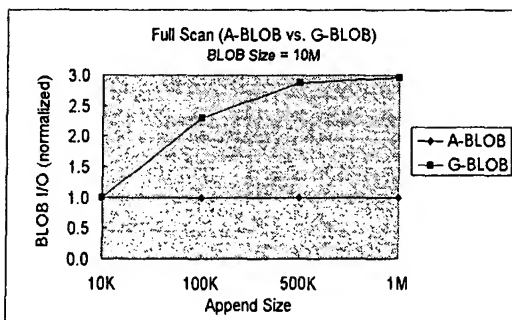
그림 8 A-BLOB과 G-BLOB의 첨부 I/O 비용 비교

A-BLOB과 G-BLOB의 전체 검색 I/O 비용은, 10M 바이트 크기의 첨부 전용 BLOB에 대하여 첨부부의

크기 (즉, 한번에 첨부되는 데이터의 크기)를 10K, 100K, 500K, 1M 바이트 등으로 변화시키며 각각에 대하여 측정하였다. 다시 말해서 BLOB의 크기가 10M 바이트가 될 때까지 한번에 10K 바이트 (100K, 500K, 1M 바이트) 씩 첨부하여 생성된 BLOB에 대한 전체 검색 I/O 비용을 구하였다. 첨부 크기가 비교적 작을 때는 A-BLOB과 G-BLOB 모두 전체 검색 비용이 높지만 (특히, 첨부 크기가 A-BLOB의 최소 세그먼트 크기인 16K 바이트보다 작을 때는 A-BLOB과 G-BLOB의 전체 검색 비용이 동일함을 보이고 있다.) 첨부 크기가 점차 커질수록 G-BLOB의 전체 검색 비용은 감소하지 않는데 반하여 A-BLOB의 전체 검색 비용은 감소함을 나타내고 있다 (그림 9(a)). 첨부 전용 BLOB을 A-BLOB에 저장하는 경우에 비하여 G-BLOB에 저장하면 전체 검색 비용이 첨부 크기가 증가해 감에 따라 약 3배에 가까워 짐을 보여준다 (그림 9(b)).



(a)

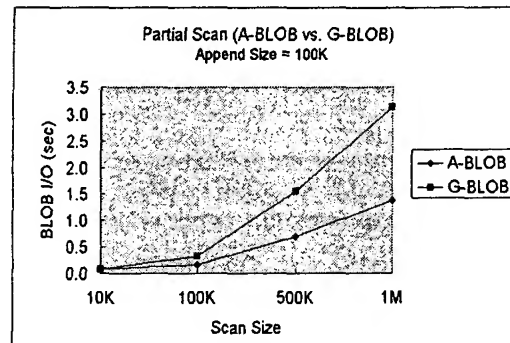


(b)

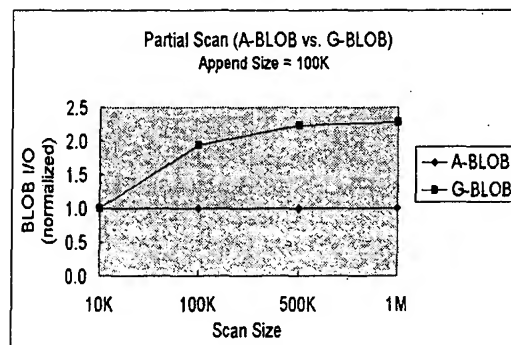
그림 9 A-BLOB과 G-BLOB의 전체 검색 I/O 비용 비교

A-BLOB과 G-BLOB의 부분 검색 I/O 비용은, 한번에 100K 바이트씩 첨부하여 생성된 10M 바이트 크기의

첨부 전용 BLOB에 대하여 검색의 크기 (즉, 검색되는 바이트 범위의 크기)를 10K, 100K, 500K, 1M 바이트 등으로 변화시키며 각각에 대하여 측정하였다. 각 검색 크기에 대하여 무작위로 100개의 바이트 범위를 생성하여 해당 데이터를 검색하고 그 I/O 비용을 측정하여 평균값을 구하였다. 검색 크기가 비교적 작을 때는 A-BLOB과 G-BLOB 모두 부분 검색 비용이 낮지만 검색 크기가 점차 커질수록 부분 검색 비용이 증가함을 나타내고 있다. 그런데 A-BLOB의 경우 그 증가폭은 G-BLOB의 그것에 비하여 완만함을 보이고 있다 (그림 10(a)). 첨부 전용 BLOB을 A-BLOB에 저장하는 경우에 비하여 G-BLOB에 저장하면 부분 검색 비용이 검색 크기가 증가해 감에 따라 약 2.5배에 가까워 짐을 보여준다 (그림 10(b)).



(a)



(b)

그림 10 A-BLOB과 G-BLOB의 부분 검색 I/O 비용 비교

5. 결 론

본 논문에서는 관계 데이터 모델 하에서 멀티미디어

데이
ject.
범용
로 다
이타
미디
설계
티미
전에
이 다
검증
현
용 I
재하
채,
제공
한 것
BLOB
티미
의 특
만족
것으
BI
성능
BLOB
으로
들 수

[1]

[2]

[3]

[4]

[5]

[6]

데이터를 저장하기 위한 BLOB (Binary Large Object) 타입에 대하여 연구하였다. 멀티미디어 DBMS가 범용성을 갖는 단일 BLOB 타입만을 제공함으로써 서로 다른 특성과 처리 요건을 갖는 다양한 멀티미디어 데이터들을 모두 이 BLOB 타입에 저장하는 것보다, 멀티미디어 데이터의 여러가지 특성과 처리 요건을 고려하여 설계된 다양한 BLOB 타입들을 제공함으로써 어떤 멀티미디어 데이터를 저장하는 데 있어 그 특성과 처리 요건에 가장 적합한 BLOB 타입을 선택하여 저장하는 것이 더 효율적임을 제시하고 이를 구현과 실험을 통하여 검증하였다.

현재까지 BLOB 타입을 지원하는 프로토타입 및 상용 DBMS들은, 서로 다른 멀티미디어 응용들 간에 존재하는 여러 특성과 처리 요건의 차이를 고려하지 않은 채, 모든 BLOB을 저장하기 위한 단일 BLOB 타입을 제공하고 있다. 미래의 DBMS에서는, 본 논문에서 제시한 것과 같이 읽기 전용 BLOB 타입이나 첨부 전용 BLOB 타입과 같이, 범용성은 갖추지 못했으나 실제 멀티미디어 데이터베이스 응용에서 널리 쓰이는 데이터들의 특성과 처리 요건을 단일 BLOB보다 더 효율적으로 만족할 수 있는, 다양한 BLOB 타입들을 여럿 제공할 것으로 기대된다.

BLOB 타입에 대한 향후 연구 과제로는, 본 논문의 성능 평가에 사용된 읽기전용 BLOB 또는 첨부전용 BLOB 타입 외에 실제 멀티미디어 응용들에서 효율적으로 활용될 수 있는 특성화된 BLOB 타입들의 연구들을 들 수 있다.

참 고 문 헌

- [1] T. Shetler, "Birth of the BLOB," BYTE, Feb. 1990, pp. 221-226.
- [2] P. Cotton et. al., "Supporting Large Object Strings in SQL," ISO/IEC JTC1/SC21/WG3 DBL MUN-166, Nov. 1993.
- [3] A. Biliris, "An Efficient Database Storage Structure for Large Dynamic Objects," Proc. Int'l Conf. on Data Engg., 1992, pp. 301-308.
- [4] R. Haskin and R. Lorie, "On Extending the Functions of a Relational Database System," Proc. ACM SIGMOD, June 1982, pp. 207-212.
- [5] H. Chou et. al., "Design and Implementation of the Wisconsin Storage System," Software Practice and Experience, Vol. 15, No. 10, October 1985, pp. 943-962.
- [6] M. Carey et. al., "Object and File Management in the EXODUS Extensible Database System,"

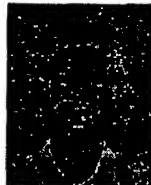
Proc. Int'l Conf. on VLDB, 1986, pp. 91-100.

- [7] T. Lehman and B. Lindsay, "The Starburst Long Field Manager," Proc. Int'l Conf. on VLDB, August 1989, pp. 375-383.
- [8] A. Biliris, "The Performance of Three Database Storage Structures for Managing Large Objects," Proc. ACM SIGMOD, 1992, pp. 276-285.
- [9] 이규철, 강현철, 박영철, "바다 DBMS 기능 확장에 관한 연구," 한국전자통신연구소, 1차년도 최종 보고서, 1992년 12월.
- [10] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques," Morgan Kaufmann publishers, 1994.



최 기 호

1992년 중앙대학교 전자계산학과 졸업(공학사), 1995년 중앙대학교 컴퓨터공학과 대학원 졸업(공학석사), 1995년 ~ 현재 한국통신기술 재직. 관심분야는 멀티미디어 데이터베이스, 분산 데이터베이스.



강 은 지

1994년 중앙대학교 전자계산학과 졸업(공학사), 1994년 ~ 현재 중앙대학교 컴퓨터공학과 대학원 석사과정 재학중. 관심분야는 클라이언트-서버 DBMS, 멀티미디어 데이터베이스, 트랜잭션 처리.



강 현 철

1983년 서울대학교 컴퓨터공학과 졸업(공학사), 1985년 U. of Maryland at College Park, Computer Science(M. S.), 1987년 U. of Maryland at College Park, Computer Science(Ph. D.), 1988년 ~ 현재 중앙대학교 컴퓨터공학과 부교수. 관심분야는 클라이언트-서버 DBMS, 분산 데이터베이스, DBMS 저장시스템.